TFG

Release 1.0

Hugo Ferreira

Jun 22, 2019

CONTENTS

Contents:

CHAPTER

ONE

KEYWORDS DOCUMENTATION!

class keywords.**OrderedDict**
OrderectDict class

Creates a new Dictionnary data structure that allows multiple append on the same key.

Args: Dict :The data structure Dictionnary.

keywords.**concatenate_candidates_grampal**(*graph*, *nodes*, *text*)
Get the multiwords from the top nodes of the graph using spacy as service.

Args: graph (*igraph*): Graph to be analyse.

nodes (list): The list of top nodes.

text (str): Text of origin.

Returns: nodes (list): The list of the multiwords.

keywords.**concatenate_candidates_spacy**(*graph*, *nodes*, *text*)
Get the multiwords from the top nodes of the graph using spacy as service.

Args: graph (*igraph*): Graph to be analyse.

nodes (list): The list of top nodes.

text (str): Text of origin.

Returns: nodes (list): The list of the multiwords.

keywords.**create_graph_grampal**(*text*, *k=2*)
Create a graph with the keywords and their links using grampal as service.

Args: text (str): The text of origin.

k (int): The correlation value ,by default = 2.

Returns: graph (*igraph*): The graph generated.

keywords.**create_graph_spacy**(*text*, *k=2*)
Create a graph with the keywords and their links using spacy as service.

Args: text (str): The text of origin.

k (int): The correlation value ,by default = 2.

Returns: graph (*igraph*): The graph generated.

keywords.**custom_tokenizer**(*nlp*)
Redefine the custom tokenizer of spacy .

Args: nlp (*nlp*): The tokenizer from spacy.

Returns: nlp (*nlp*): The new custom tokenizer.

keywords.**main**(*arguments*)
Main function of the keywords module.

Kwargs: graph (*igraph*): The graph to be printed.

path (str): The path.

keywords.**pagerank**(*graph*)
Use the Google's pagerank algorithm to set a value for each node.

Args: graph (*igraph*): Graph to be analyse.

Returns: values (list): The list of values generated.

keywords.**print_graph**(*graph*, *path*)
Print the graph generated, it was used for validation on small graph, currently unused .

Args: graph (*igraph*): The graph to be printed.

path (str): The path.

keywords.**sort_occurences**(*graph*)
Get an array of the nodes sorted_by_occur by occurence.

Args: graph (*igraph*): Graph to be analyse.

Returns: nodes (list): The list of nodes generated.

keywords.**sort_values**(*graph*)
Get an array of the nodes sorted_by_occur by value.

Args: graph (*igraph*): Graph to be analyse.

Returns: nodes (list): The list of nodes generated.

keywords.**topnodes**(*graph*, *top*)
Extract the Top nodes with higher values.

Args: graph (*igraph*): Graph to be analyse.

Top (int): Number of nodes we want to get from the top.

Returns: nodes (list): The list of nodes generated.

CHAPTER

TWO

GRAMPAL WS DOCUMENTATION!

class ws.**Grampal**(*service=None*)

Grampal service class

This class implements all the functionality of the Grampal ws, allowing the tokenize and analyse of a phrase

analiza(*phrase*)

Analyse a phrase using Grampal's service

Args: phrase (str): The phrase to be analyse.

Returns: Object: The request object if successful, *None* otherwise.

The status_code of the response can be checked:

{ '200': 'success', '404': 'not found'

}

analiza_get(*phrase*)

GET function of the Grampal service

Args: phrase (str): The phrase to be analyse.

Returns: Object: The request object if successful, *None* otherwise.

The status_code of the response can be checked:

{ '200': 'success', '404': 'not found'

}

analiza_post(*phrase*)

POST function of the Grampal service

Args: phrase (str): The phrase to be analyse.

Returns: Object: The request object if successful, *None* otherwise.

The status_code of the response can be checked:

{ '200': 'success', '404: 'not found' }

info_lemma(*phrase*)

Parse the response from the Grampal ws extracting the lemma information

Args: phrase: Phrase to be analyse

Returns: String: The lemma information if successful, *None* otherwise.

info_orig(*phrase*)

Parse the response from the Grampal ws extracting the word of origin

Args: phrase: Phrase to be analyse

Returns: String: The word of origin of the token

info_syntactic(*phrase*)
Parse the response from the Grampal ws extracting the syntactic information

Args: phrase: Phrase to be analyse

Returns: String: The syntactic information if successful, *None* otherwise.

CHAPTER

THREE

CREATE_JSON DOCUMENTATION!

create_json.**multiple_json**(*file_name*)
Function that creates multiple json (one for every row) from the babelnet index format

Args: file_name: (str): The name of the index file.

create_json.**single_json**(*file_name*)
Function that creates a single json from the babelnet index format

Args: file_name: (str): The name of the index file.

CHAPTER

FOUR

ELASTIC_BULK DOCUMENTATION!

elastic_bulk.**decode_nginx_log**(*_nginx_fd*)
Function that parse the source information from a json.

Args: _nginx_fd (str): The name of the json file.

Returns: Object: The json object generated

elastic_bulk.**es_add_bulk**(*nginx_file*)
Function that bulk the information from a json.

Args: nginx_file (str): The name of the json file.

elastic_bulk.**main**()
Main function of the elastic_bulk module.

CHAPTER

FIVE

ELASTIC DOCUMENTATION!

elastic.**concept_extraction**(*es*, *list_keywords*, *mode*, *model*, *reach=10000*)
Extract the main concepts of a list of keywords

Args: es (*elasticsearch*): The elasticsearch service.

list (`str`): The list of keywords extracted generated in keywords.py mode (`int`): The mode of retrieval(0 for direct reference,1 for aparition in others lemma model (`int`): The model of the increse(0 linear,1 exponential) reach (`int`): The limit of keywords to be check

Returns: search (*search object*): The results of the search function.

elastic.**extract_list**(*es*, *list_keywords*, *mode*, *model*, *reach=10000*)
Extract the most representative concept of each keyword

Args: es (*elasticsearch*): The elasticsearch service.

list (`str`): The list of keywords extracted generated in keywords.py mode (`int`): The mode of retrieval(0 for direct reference,1 for aparition in others lemma model (`int`): The model of the increse(0 linear,1 exponential) reach (`int`): The limit of keywords to be check

Returns: search (*search object*): The results of the search function.

elastic.**insert_concept_results**(*es*, *file*, *index_name*, *data*)
Insert the results of the top concept representation

Args: es (*elasticsearch*): The elasticsearch service.

file (`str`): The name of the file to be analyze index_name (`str`): The name of the index to insert the data data (*search*): The response of the search function

Returns: bool (*index*): The index object with the response of the index function

elastic.**insert_list_results**(*es*, *file*, *index_name*, *data*)
Insert the results of the individual concepto extraction

Args: es (*elasticsearch*): The elasticsearch service.

file (`str`): The name of the file to be analyze index_name (`str`): The name of the index to insert the data data (*search*): The response of the search function

Returns: bool (*index*): The index object with the response of the index function

elastic.**search_results**(*es*, *file*)
Check if they are results matching the text and return the results

Args: es (*elasticsearch*): The elasticsearch service.

file (`str`): The name of the file to be analyze

Returns: bool (`boolean`): True if match, otherwise None object is returned

PYTHON MODULE INDEX

INDEX